# Synthesizing Pareto Optimal Decision for Autonomic Clouds using Stochastic Games Model Checking

Azlan Ismail
Faculty of Computer and
Mathematical Sciences
Universiti Teknologi MARA
Shah Alam, Selangor, Malaysia
Email: azlanismail@tmsk.uitm.edu.my

Marta Kwiatkowska
Department of Computer Science
University of Oxford
Oxford, UK
Email: Marta.Kwiatkowska@cs.ox.ac.uk

***Abstract*—The ability to automatically generate and guarantee the optimal decision for self-adaptation is important especially when there are multiple quality objectives that need to be satisfied, the uncertainties in the adaptation outcome, and the time-varying resource demands, especially in the autonomic cloud systems. To address this issue, in this paper, we propose an approach to automatically encode the adaptation decision behavior and the multiple quality objectives, as well as synthesizing the behaviour to fulfill the specified objectives. In the approach, we emphasize the relation between quality objectives expressed as a variant of temporal logic specification and the domain-specific Service Level Agreements (SLA) (i.e. cloud environment). The approach also covers the abstraction method for representing the adaptation behavior as stochastic games, and the re-synthesis method to adjust the threshold values, if failing to satisfy the predefined thresholds. We apply the stochastic games model checking with strategy synthesis to realize the approach. The Pareto-set computation is utilized to support the adjustment of threshold values. We present a set of validation results to show the effectiveness and performance of the proposed approach.**

***Index Terms*—autonomic clouds, multiple quality objectives, strategy synthesis, stochastic games model checking.**

## I. Introduction

Self-adaptation is increasingly used to address the uncertainty and complexity of modern software systems and their environments [8], [11], [18], [24]. Self-adaptive (or *self-managing*) software continually monitors and analyzes its behaviour and that of its environment, synthesizing and executing reconfiguration plans when needed to avoid violating system goals [17]. This monitor-analyze-plan-execute (MAPE-K) closed control loop [12] has been successfully used in a wide range of software systems.

Self-adaptive capability has been promoted to enable autonomic clouds environment [20]. The goal is to allow the cloud to self-managed its own environment. This is important to address the challenges in managing cloud resources in a complex, large-scale, and heterogeneous distributed environment. In addition, the cloud can collaborate with other clouds (also known as cloud collaborators) in terms of sharing resources in provisioning cloud applications. The need for having a collaborator comes from the limitation of the cloud itself. For instance, the research centres of the University might have limited resources in executing complex large-scale scientific applications. Thus, they can collaborate with the industry research centres to enable shared resources.

One of the challenges in the autonomic cloud is to find the optimal cloud collaborator to deploy and execute the respective cloud application. As there are more than one potential cloud collaborators in the group (also called as cloud ensemble) [13], the cloud in need (also called as cloud requestor) has to decide which of them is the best to perform the deployment and execution task. One of the important information to make decision is to consider the cloud collaborator reputation [4]. However, more information is required to improve the quality of the decision.

In this work, we pay a specific attention to improving the effectiveness of the self-adaptation decision by focusing on the issues influencing the generation of optimal adaptation plan. We view the issues from three perspectives, the multiple dimension of *quality objective*, the *resource variation* and the *uncertainty* of the system environment.

In real setting, there is a need for the self-adaptation to fulfill multiple dimension of quality objectives imposed by the application user. The common approach to measure the quantitative information towards satisfying multiple quality objectives is based on weighted aggregation method [2], [3], [23]. However, in certain situation, some information is unknown or not provided, in particular, the weights or preferences of each quality objective. Assuming such information is always provided limits the effectiveness of the adaptation.

Furthermore, in the cloud environment, the demand for resources can be varied overtime, and it depends on each cloud in controlling its own resource. Assuming the resources are partitioned in terms of a series of time region, the cloud can decide to deploy the requested cloud application in one of the region. Without exposing this information, the cloud requestor might have selected a cloud collaborator which can potentially violate certain constraint, such as a deadline (if

exist). Meanwhile, the uncertainty of the system environment refers to the uncertainty of commitment of the cloud in the cloud ensemble. This kind of commitment is expected especially in a voluntary setting [19]. Failing to consider both aspects, the resource variation and the uncertainty of commitment can also affect the effectiveness of the decision. To the best of our knowledge, the related work only considers the latency of the adaptation [2], which is insufficient for the resource deployment problem.

Therefore, this research aims to propose a decision-making approach to generate an optimal adaptation plan which aims to tackle the above issues to improve the effectiveness of the self-adaptation decision for autonomic cloud environment.

For realizing the approach, we employ stochastic games model checking with multiple objective properties synthesis [6]. The technique enables (i) abstracting the decision-making behaviour from the controlled and uncontrolled perspective, as well as associating the quality attributes based on turn-based stochastic multi-player games model, (ii) specifying multiple objective properties with RPATL and its Boolean combination, (iii) computing the threshold values via Pareto-set computation, (iv) synthesizing the optimal adaptation plan via multi-objective strategy synthesis method, and (v) extracting the optimal action from the synthesis outcomes.

In this paper, we contribute a novel decision-making approach that automatically encode the model and properties specification, and synthesizes for optimal adaptation plan. The model encoding process takes several inputs, namely the number of cloud collaborator and its variation in specifying the model specification. The properties encoding process takes the inputs from the Service Level Objectives (SLOs) to automatically generate the specification. The synthesis process consists of several tasks, namely, instantiating of the model, the assignment of quantitative information, and synthesizing the optimal plan. In the case of failing to synthesize the correct plan on the first attempt, the task of adjusting the threshold values is performed with the support of Pareto set computation followed by re-synthesizing the plan.

We validate the proposed approach by utilizing the cloud application deployment scenario to show its effectiveness. We also provide some performance results to illustrate its scalability.

The rest of this paper is organized as follows. Section II introduces the adaptation context for this research. Section III provides the background information of the applied technique. In Section IV, we present the proposed approach and, in Section V, we provide the validation results. We highlight related work in Section VI and conclude the paper in Section VII.

## II. Adaptation Contexts

This section introduces the self-adaptation architecture which becomes our referred model, the autonomic cloud and the adaptation decision scenarios.

### A. Self-adaptation Architecture

The proposed approach discussed in this paper can be mapped into the common self-adaptation architecture based on MAPE-K which emphasizes the monitor-analysis-planning-execution phases. In general, the monitoring is needed to observe the condition of the managed system. The analysis is used to perform some predictions or diagnose any problem. The planning is required for finding the appropriate adaptation plan. The execution is meant to carry out the plan on the managed systems. Meanwhile, knowledge repository is used to support the phases. The MAPE-K cycle is performed by a component known as autonomic manager (AM).

There are two common adaptation approaches, reactive and proactive approach [14]. In reactive approach, the adaptation is triggered when the actual problem has occurred and detected. Meanwhile, in proactive approach, the adaptation is triggered by looking ahead to predict when to adapt. As we are concerned with the planning phase, which needs to decide the best adaptation action, in general, the proposed approach can be applied into reactive and proactive adaptation. However, synthesizing for optimal solution is more significant to support proactive adaptation since it consumes more time and resource as compared to the rapid approach, i.e. incremental selection [27].

### B. Autonomic Cloud Environment

We utilize the autonomic cloud environment to illustrate the proposed approach. For this reason, we refer to the Science Cloud Platform (SCP) within the ASCENS project [20].

A SCP is a computing node or simply a cloud. An instance of SCP comprises an AM which enables the cloud to adjust its resource autonomously and proactively, in relation to the network resilience, and resource failures.

The autonomic cloud environment can contain multiple clouds that work collaboratively in sharing their resources (voluntarily basis) to provision the cloud applications [19]. Any cloud can join or leave the collaboration, also known as ensemble.

As an AM, each cloud can take several roles in the ensemble, based on Helena approach [13]. The roles are to store and deploy a cloud application (deployer), to find a cloud collaborator that can execute the cloud application (initiator), to execute the cloud application (executor), and to observe the execution of the application (monitor).

### C. Adaptation Decision Scenario

In this paper, we focus on the optimal decision making problem of self-adaptation by a cloud. Specifically, in the autonomic cloud, the adaptation decision can be performed at different cloud role. Thus, we limit our work to the *Initiator* role, in particular, when the cloud realizes that it cannot execute the application due to its limited resources and requires to finding the optimal cloud collaborator that can do the job. The solution to the decision-making problem is an optimal decision path (i.e. adaptation plan) of choosing a cloud collaborator and its consequence outcome.

Figure 1 illustrates the application deployment scenario. There are 8 computing nodes within the same ensemble which are willing to share their resources and disclose the relevant

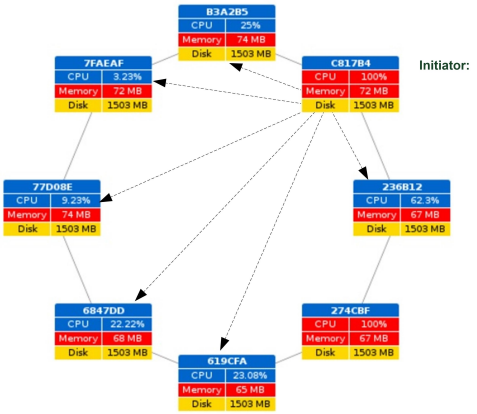


Fig. 1. Application Deployment Scenario in Autonomic Clouds

information. Each computing node has AM capability. Any computing node within ensemble can receive an application provisioning request, deploy the application, or even finding the right computing node for deploying the application. There are a set of qualitative objectives associated to the application, namely, the cost, response time and reliability.

In this case, assuming computing node *C817B4* has received a request to deploy an application with multiple quality objectives. However, at that point of time, it is not able to deploy the application due to the resource constraint. As a result, it turns itself becoming as an Initiator which needs to find the cloud collaborator to deploy and execute the application.

### D. Crucial Factors Affecting the Decision

We are concerned with several crucial factors that affect the effectiveness of the adaptation decision, explained as follows:

*1) Absence of weight information:* The weights have been utilized to enable multiple quality aggregation for the self-adaptation decision [9]. Each weight is meant for a single quality objective (e.g. $0.6$ for cost, $0.4$ for time). By having this information, multiple quality objectives can be reduced a single objective. However, the *absence of weight* can occur especially when the users are not able to provide the required weights. This situation will affect the ability to find the decision.

*2) Time-varying resource workloads:* The *time-varying resource workloads* of the potential cloud collaborators can be viewed as a series of time periods with different estimated resource utilization. Example of time-varying resource workloads are high utilization and low utilization period [21].

These periods are significant to the clouds in distributing their workloads to achieve a load balanced. A cloud may receive a request during its high utilization period. Without considering this factor in deciding its commitment to collaborate with others (i.e. willing to deploy and execute the requested application), it will simply reject the request due to unable to cater the requested application. In fact, there is a high probability that the performance of the application is significantly degraded. However, this behaviour will not benefit the collaborative cloud environment, as in ensemble, although it has its own goal. Thus, by considering this factor, it may be willing to commit to collaborate, subject that the cloud requester can bear with some delays, whilst maintaining the expected quality objectives of the application.

From the cloud requestor perspective, this information can assist them to be aware about the commitment of their collaborators, especially when dealing with the actual execution. This is important since there could a deadline constraint imposed to the application. Otherwise, delaying the execution of the application on other cloud could be the best option (i.e. maintaining the expected quality), assuming the cloud requestor itself is dealing with longer high utilization period, or unable to commit in ensemble in further period. In this context, we assume a potential cloud collaborator is willing to share the information related to these periods, but not their internal goals. Thus, the cloud requestor can make an effective decision in choosing its potential collaborator.

*3) Uncertainty of commitment:* The *uncertainty of commitment* refers to the commitment of the cloud collaborator in the ensemble. This is important since the participation of the clouds in ensemble is a voluntary basis. There is possibility where the selected cloud collaborator leaves the ensemble before completing the execution of the application. Failing to consider this factor can affect the effectiveness of the adaptation.

## III. Stochastic Games Model Checking with Strategy Synthesis

Stochastic games provide a convenient abstraction for modelling the behavior of systems that comprise several players that may either compete or cooperate in order to achieve a certain goal, while also exhibiting probabilistic behavior [26]. In this paper, we consider zero-sum, turn-based stochastic multiplayer games (SMG) [6]. In a turn-based game players move in turns, i.e. there is exactly one player move (i.e. to make the choice of action) at each step. A zero-sum game is a game where two players have directly conflicting objectives and the payoff of one player complements the payoff of the other. The goal of the game from a player perspective is to win the game, that is, to determine the optimal choice. The formal definition is as follows:

*Definition 3.1:* SMG A (turn-based) stochastic multi-player game (SMG) is a tuple $G$, where $\Pi$ is a finite set of players, $S$ is a finite, non-empty set of states, $A$ is a finite, non-empty set of actions, $(S_i)_{i\in\Pi}$ is a partition of S, $\Delta : S \times A \to D(S)$ is a (partial) transition function that maps state-action pairs to probability distributions over $S$, $AP$ is a finite set of atomic propositions, $\chi : S \to 2^{AP}$ is a labelling function assigning to each state a set of atomic propositions taken from a set $AP$, and $r$ is a reward structure mapping each state to a non-negative rational reward, $r : S \times A \to \Re$.

In an SMG, for each state $s \in S$, an action is chosen by a single player $i \in \Pi$ that controls the state $s \in S_i$. The action is selected from a set of actions for the state, $A(s)$. Once a player has made the choice, the successor state is chosen

according to probability distribution $\Delta(s, a)$. The selection of action from a state to the next state creates a path. A path of $G$ is a possibly infinite sequence $\lambda = s_0 a_0 s_1 a_1 ...$ such that $a_j \in A(s_j)$ and $\Delta(s_j, a_j)(s_{j+1}) > 0$ for all $j$.

An SMG consists of two types of states, player states and probabilistic states. Players may have several nondeterministic choices available in non-probabilistic states, from which they select a transition to move to the next state. A *strategy* in an SMG is a way to resolve non-determinism for each player. In probabilistic states transitions are given as probability distributions. An SMG is unfolded into *paths*, that is, sequence of states which alternate between player states and probabilistic states. A *transition trace* comprises a sequence of action labels of a path. An SMGs can be annotated with *rewards* that can be used to formulate a variety of quantitative analyses, e.g. of the accumulated cost along specific paths.

### A. Properties

The properties of SMGs are expressed in the probabilistic temporal logic RPATL [6]. Syntactically, RPATL is a CTL-style branching-time temporal logic that distinguishes between path formulae ($\psi$) and state formulae ($\phi$). An example of a path formula is $\mathbf{F}\phi$ (eventually $\phi$). RPATL can be used to formally specify that a coalition of players $\langle\langle P\rangle\rangle$ has a strategy which can ensure that the probability of an event's occurrence or that an expected reward measure meets some threshold, irrespective of the actions of the other players. The two types of properties are expressed using the probabilistic operator $\mathbf{P}$ and the reward operator $\mathbf{R}$, respectively, and the threshold can be replaced with '=?' to require the computation of the actual probability of reward. For instance, the evaluation of the RPATL property $\langle\langle P\rangle\rangle\ \mathbf{R}\{r\}_{\max=?}\ [\mathbf{F}\phi]$ yields the *maximum accumulated reward $r$ achievable by the players in coalition $P$ by using a strategy which ensures that a state satisfying formula $\phi$ is reached independently of the strategies of other players*. RPATL can be used to reason about the total reward $[\mathbf{C}]$, mean payoff $[\mathbf{S}]$ or long-run ratio of two rewards. As we are concerned with the total reward, the formulated games must be stopping, which have terminal states with zero reward that can be reached almost surely under all strategies.

There are two types of properties, single objective and multi-objective properties. Classical RPATL is used to state a single objective property, whilst multi-objective is a Boolean combination of single objective properties. The multi-objective properties must be of the same type (i.e. either all probabilistic or reward-based). An example of multi-objective is $\langle\langle P\rangle\rangle\ (\mathbf{R}\{r_1\}_{<x}\ [\mathbf{C}]\ \&\ \mathbf{R}\{r_2\}_{>y}\ [\mathbf{C}])$. This property expresses that *the players in coalition $P$ ensure a strategy where the accumulated reward $r_1$ less than $x$ is guaranteed AND the accumulated reward $r_2$ greater than $y$ is guaranteed, independently of the strategies of other players*.

### B. Strategy Synthesis

A strategy for a player resolves non-determinism of player choices (i.e. choosing which action to take) from the initial state until the terminal state is reached in order to satisfy the specified properties. Strategy synthesis aims to find the winning strategy for a player. The synthesis algorithm used to find the winning strategies is based on the *value iteration* method [5] implemented in the PRISM-games tool [16].

There are several types of strategies resolve non-determinism [6]. The *memoryless deterministic strategies* are generated to ensure the satisfaction of single objective properties. On the other hand, *stochastic memory update strategies*, which utilize explicit memory representation, are generated to satisfy multi-objective properties.

While optimal strategies can be constructed for single objective properties, for multi-objective properties optimal strategies may not exist. Hence, existing solutions for multi-objective require the computation of $\varepsilon$-approximations of Pareto sets and the corresponding $\varepsilon$-optimal strategies [7]. The Pareto set contains a set of optimal achievable vectors of reward bounds, also called Pareto vectors, which represent *optimal trade-offs* between objectives in an MO property. The Pareto set is computed using an algorithm based on the value iteration algorithm.

## IV. Adaptation Decision Approach

Our approach aims at improving the effectiveness of the adaptation decision when dealing with multiple quality objectives, the absence of weights, the time-varying resource workloads, and the uncertainty of commitment of the potential cloud collaborators. The decision making is made whenever an adaptation is anticipated, especially in the context of i.e. proactive adaptation [23]. This also means, the adaptation decision is made before the actual problem occurs.

The important steps towards automatically deciding the optimal adaptation is identified as three stages. Firstly, encoding the adaptation decision behavioural model, and the multiple quality objective properties specification. In general, the encoding can be done online and offline [1]. The initial specification can be encoded offline, whilst the online encoding can be done if the structure (i.e. number of adaptation action, number of variation) is changed. Secondly, pre-processing the encoded models followed by synthesis or re-synthesis process in the case of failure. Thirdly, extracting the optimal action, in this case, the best cloud collaborator from the synthesis outcomes. The details are explained as follows.

### A. Encoding Multiple Quality Objectives

We propose an automated encoding of multiple quality objectives based on predefined rules. The rules map the elements of SLA contract i.e. CSLA [25] into the temporal logic specification template.

There are a few segments described in the SLA contract. We are concerned with the Service Level Objectives (SLO) segment that consists of a set of quality elements including the quality objective parameter, its comparator, and its threshold. We use three objective parameters to illustrate this approach, namely, the response time, cost, and reliability. A sample segment of the response time is shown in Figure 2.

```
</csla:objective id="responsetime" .....>
<csla:expression metric="rt" comparator="lt"
threshold="3" value="100"......>
```

Fig. 2. Sample of SLO in SLA

The temporal logic specification template is based on RPATL. The coalition of the properties focus on the controlled decision maker, in our context, player 1 as explained in IV-B. Furthermore, we use a single $\mathbf{R}$ operator to represent a single quality objective in SLO (e.g. response time). We also use the accumulated reward reasoning $[\mathbf{C}]$. For multiple quality objectives, the $\mathbf{R}$ operator is connected with the Boolean combination. In this approach, we limit to AND $(\&)$ only. Example of the specification structure is as follows:

$$\langle\langle p1\rangle\rangle\ (\mathbf{R}\{r_0\}_{\bowtie y_0}\ [\mathbf{C}]\ \&\ \mathbf{R}\{r_1\}_{\bowtie y_1}\ [\mathbf{C}]) \tag{1}$$

Based on this structure, the objective parameter in SLO is mapped to reward attribute $r_i$ which is expressed in the behavioral model and is also used to define the bound attribute $y_i$, and the comparator in SLO (i.e. "lt" is equivalent to less than) is used to denote $\bowtie$. Meanwhile, the threshold value in SLO will be assigned into attribute $y_i$, as discussed in Section IV-C.

### B. Encoding the Decision Behavior

Our approach provides an automated encoding for abstracting the adaptation decision behaviour model which can be done online, and based on some predefined structure. Encoding the model offline is viable if all the structure of the model is assumed to be fixed for any adaptation cycle. However, in certain situation, this is insufficient, especially if the potential adaptation action can change i.e. the potential cloud collaborator to be selected.

To enable the automated encoding, some elements of the specification are predefined, whilst others are dynamically created, based on two main information, namely, the number of cloud collaborator and the number of variation for each cloud. The number of potential cloud collaborator can be varied since the cloud can join and leave the cloud ensemble [13](a collective of clouds that is formed based on demand) at their preferred time. The number of variation can also be varied depending on how fine or coarse of the time variation. In our approach, we assume that both numbers are deterministic can be obtained from the knowledge repository.

The core elements of the adaptation decision model that need to be encoded (using PRISM language) are classified into four aspects.

Firstly, we encode *the players* of the stochastic games that abstractly represent the decisions involved. In the context of application deployment scenario, we are concerned with two perspectives of decisions:

- The *controlled decision making* that aims to select the best adaptation action. This is equivalent to select which cloud collaborator to deploy and execute the cloud application. We can also view this decision as the cloud requestor perspective. For abstracting the decision behavior in the model, we use a player state also called as player 1, $p1$.
- The *uncontrolled decision making* which aims to choose the time region. This decision is supposedly made by the potential cloud collaborator and thus, it is beyond the control of the cloud requestor. Thus, for the abstraction purpose, we use another player state, also called as player 2, $p2$. Modeling this decision as another player represents the worst-case scenario from $p1$'s perspective.

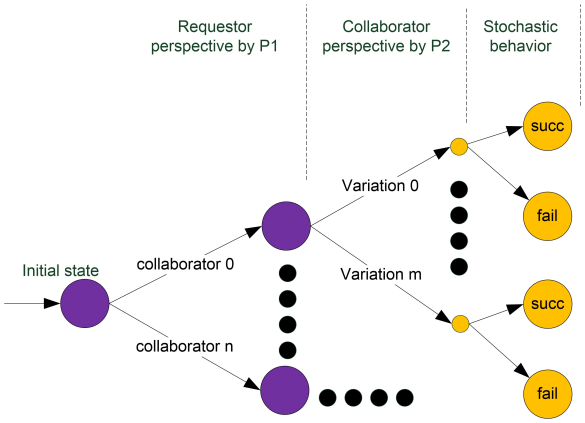


Fig. 3. Adaptation Decision Behavioral Model

Figure 3 shows the behaviors of the players for both decisions. Example of the encoded player definition is as follows:

```
player p1
decideCollab [n0], [n1]
endplayer
player p2
environment [n0e0], [n0e1], [n1e0], [n1e1], [n1e2]
endplayer
```

Secondly, we encode *the decision logic of p1* that needs to select a potential cloud collaborator $[n_i]$. Technically, $p1$ makes a transition from the state of having no selection to the state of choosing one collaborator as illustrated in Figure 3. A single decision logic is meant for abstracting a single selection. Thus, the number of required decision logic to be encoded depends on the number of potential cloud collaborator that joins the ensemble. Assume cloud $n_i$ is a potential collaborator that can be chosen, the decision logic can be encoded as follows:

```
.....
[n_i] (t=0) & (goal=false) & → & (n'=i) & (t'=1);
.....
```

Thirdly, we encode *the decision logic of p2* that resolves the non-determinism of time variation of the cloud collaborator $[n_i e_j]$. This behavior is viewed as the uncontrolled decision from player 1 perspective. The number of variation $e_j$ can be varied depending on the time slots imposed by each cloud

collaborator $n_i$. Thus, we associate a resource constraint to check the feasibility to execute the cloud application at a specific time variation. This constraint is important to reduce the search space for making the selection. To formulate the resource constraint, we consider three parameters, namely, the estimated resource usage of the application $app_{rs}$, the estimated resource utilization for each time variation $var_{rs}^{i,j}$ and the maximum utilization $mx_{rs}$.

In addition, we associate the stochastic states with $p2$, to represent the uncertainty effect of the cloud commitment in the ensemble (i.e. continue participating, leaving), which results in either success or failure of the adaptation. Example of the encoded decision logic for a variation $e_j$ of a cloud $n_i$ is formulated as follows:

$$
\begin{array}{l}
\ldots\ldots\ldots \\
[n_i e_j]\ (t{=}1)\ \&\ (n{=}i)\ \&\ (app_{rs}{+}var_{rs}^{i,j}{\leq}mx_{rs})\ \&\ \rightarrow \\
rel_i^j : (goal'{=}\texttt{true})\ \&\ (t'{=}0) + 1{-}rel_i^j : (goal'{=}false)\ \&\ (t'{=}0); \\
\ldots\ldots\ldots
\end{array}
$$

### C. Multi-objective Strategy Synthesis

The automated synthesis process takes the encoded model and properties specification, and a set of quantitative information to produce the optimal plan. The successful synthesis is achieved when it can resolve the non-determinism which satisfy the multi-objective properties.

In general, the synthesis begins with pre-processing phase, which involves three steps, (i) parsing the model and properties specification into internal representation, (ii) instantiating the model instance based on explicit-state representation, and (iii) assigning values into the model and properties. After that, the synthesis is performed by using the stochastic games model checking engine with strategy synthesis capability. In the case of failure, re-synthesis phase is executed. The details are explained as follows.

*1) Pre-processing Phase:* Step (i) and (ii) can be done only once, assuming, there is no change in the future. If a change is needed, then these two steps need to be re-executed, which can be either online or offline. As discussed in the previous work, offline stage can significantly improve the performance [22].

Step (iii) is needed to assign runtime quantitative information into the model and threshold into the properties. We also note that, this step can be part of encoding process. This is applicable if the model behavior has to be re-encoded, whilst the quantitative information is known and accessible at the same time.

The focused values to be assigned into the model are the estimated quantitative information for each potential cloud collaborator and its variation. These estimated values are assumed to be produced during analysis phase of MAPE-K cycle. Meanwhile, the focused values for the properties are the thresholds, which can be obtained from the SLO.

For supporting the values assignment, we apply the mapping function that maps the quantitative parameters encoded in the model with the parameters of the estimated quantitative information. The same idea applies to the threshold assignment for the properties, where the thresholds element in SLO is mapped to the thresholds parameter in the properties specification. In general, this mapping can be expressed as a function, $assign(x, y)$ where $x$ is the parameter used in the specification, and $y$ is the parameter used in the external source.

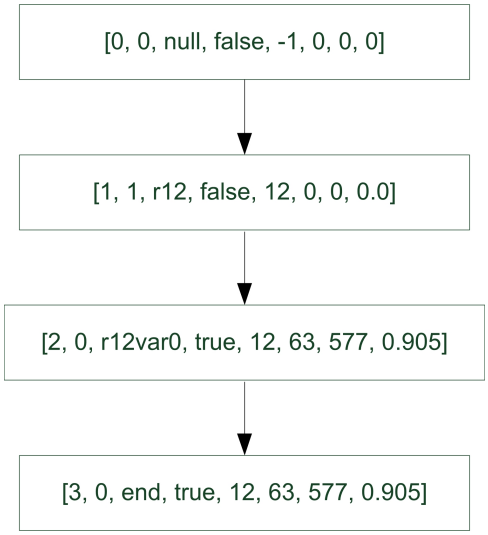


Fig. 4. Successful adaptation behavior path

*2) Synthesis Phase:* The synthesis will synthesize the model to obtain a *successful adaptation behaviour*, also called an optimal adaptation plan. The resulted successful behaviour is the one that fulfils the multi-objective properties with the predefined thresholds. Example of properties is illustrated in properties 1.

Technically, the outcome of strategy synthesis contains a set of lists where each list contains a few elements, namely, the current state, the next state, and the action to be taken. The connection between these lists which form the successful paths can be simulated using Prism games tool.

Figure 4 shows a simplified version to illustrate one of the potential paths of successful synthesis with predefined thresholds. The illustrated data is presented as $(step, turn, action, goal, i, rw\_cost, rw\_time, rw\_rel)$.

The figure is explained as follows. It begins with an initial state with step 0. This state represents the state where the adaptation is needed. From this state, player 1 makes a move (i.e. when $turn = 0$). It has a set of potential collaborators to be chosen $i = (0, ..., n)$. The next state results in a selection of collaborator with index 12 $(action = 12)$. However, at this stage, the adaptation goal is still false. Then, player 2 makes a move (i.e. when $turn = 1$) to select from a set of variation $(i_0, ..., i_m)$, which results in the selection of variation with index 0 $(action = 0)$ of collaborator 12. The estimated cost, time, and reliability associated to this selection are $(rw\_cost = 63)$, $(rw\_time = 577)$, and $(rw\_rel = 0.905)$. The outcome of this adaptation is uncertain, but is predicted to be successful. Thus, the goal has been updated to true, which ends successfully.

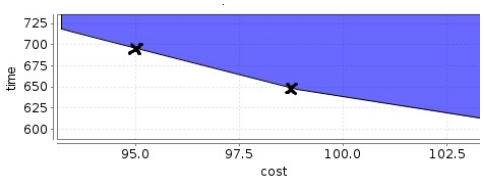


Fig. 5. Acceptable thresholds using Pareto set computation

*3) Re-Synthesis Phase:* There is a possibility where the *synthesis results in a failure* based on the predefined thresholds. In this situation, the thresholds need to be adjusted. For this reason, we utilize the Pareto set computation [7] which can determine the acceptable thresholds based on the quantitative information of the potential cloud collaborators. In the cloud scenario, the computed thresholds can be one or more pair of quality attributes, where a single pair $(x, y, z)$ refers to the upper bound for cost $x$ and time $y$, and the lower bound for the reliability $z$. If there is more than one pair of thresholds, in this work, we take the random strategy to automatically select one of them. Re-synthesizing is performed based on the selected pair. Example of visualized computed Pareto set based on cost and time is shown in Figure 5. We only show two thresholds due to the current capability of chart representation of Prism games tool. From the figure, the sample pair of the acceptable thresholds $(x, y)$ are $[93, 718]$ and $[98, 650]$.

### D. Strategy Extraction

This process is done onto the outcome of the synthesis process to obtain the optimal action, in our context, the index of the chosen cloud collaborator. To enable the extraction, it requires the transition and strategy profiles, and the identifiers of all possible cloud collaborators. The transition profile is needed since it holds all possible paths between states (i.e. the state of having no selection to the state of choosing one collaborator) together with other relevant information, namely, the action labels that include the cloud collaborator identifiers, as well as the associated quantitative information. Meanwhile, the strategy profile is required as it contains the summarized paths information which optimize multiple objective properties.

The extraction is done by matching the paths information from strategy to the transition profile. Specifically, it identifies a single path from both profiles, where the associated action label contains the identifier of one of the cloud collaborators. The matching process focuses on similarity of three elements, (i) the index of current states, (ii) the index of next states, and (iii) the index of actions. The action from the matched path is taken as the best action. In the case of more than one path matched, the best action is selected randomly. With the selected action, we can return the identifier of the cloud collaborator to the autonomic manager for the actual execution.

## V. Validation

The main purpose of our proposal is to generate the trade-off optimal plan under uncertainty with multiple quality objective, and workload variation. Thus, we provide three aspects of validations. Firstly, we evaluate its ability to recover from synthesis failure via Pareto set computation. Secondly, we show the advantage of considering variation of the cloud collaborators. Thirdly, we illustrate the performance of the proposed approach.

We begin with explaining the implementation of the simulator followed by the experimental analysis and results.

### A. Implementation

The context of the simulation only covers the aspect when a self-adaptation is needed. It simulates the decision-making process based on the proposed approach to provide the optimal strategy. It supports (i) the setting for input parameters which includes the threshold values, number of cloud collaborator and its variation, number of simulation cycle, the stage of the variable assignment (i.e. during encoding, or during pre-processing), and the need for re-synthesizing or not, (ii) the generation of quantitative information (i.e. cost, time, and reliability) of cloud collaborator profiles with a set number of variation, (iii) the execution of the proposed approach based on specific configurations, and (iv) logging the information for the analysis.

The implementation of the simulator is done using java within the eclipse environment. The main configuration to enable the development of the simulator involved two main libraries settings, namely, (i) libraries of stochastic games model checking for multi-objective properties synthesis and Pareto set computation, and (ii) additional libraries for supporting Pareto set computation, namely, Parma Polyhedra Library (PPL) [15]. Architecturally, the simulator consists of five core components, as follows:

- *Properties Generator* - This component is needed to encode properties specification as mentioned in Section IV-A. It is also used to assign values if the assignment is set to true. Otherwise, the values assignment is performed during pre-processing phase of synthesis process.
- *Model Generator* - This component is needed to encode behavioural model as discussed in Section IV-B. It is also used to assign values if the assignment is set to true. If not, the values assignment is done during pre-processing phase of synthesis process. The generation of quantitative information of cloud collaborator profiles is also part of this component.
- *Stochastic Games Planner* -This component is used to perform pre-processing, synthesizing, and re-synthesizing the behavioral model as explained in Section IV-C1, IV-C2, and IV-C3. It can also export the outcome of model checking and synthesis process which includes the transition and strategy profiles.
- *Strategy Extractor* - This component is used to to extract the optimal action that is the best cloud collaborator from the synthesis outcomes as explain in Section IV-D.
- *Synthesis Simulator* - This is the main component to control the simulation cycle that include generating the

| Cloud,Var | Init. Thresholds | Adjust. Thresholds | Syn. Time |
|---|---|---|---|
| (10,5) | [90.0, 853, 0.967] | [93.88, 1060, 0.978] | 6731ms |
| (20,5) | [99.0, 801, 0.974] | [108.26, 953, 0.980] | 10461ms |
| (30,5) | [90.0, 847, 0.975] | [80.90, 1029, 0.978] | 11860ms |
| (40,5) | [94.0, 759, 0.835] | [107.02, 944, 0.979] | 20967ms |

TABLE I
RESULTS OF RE-SYNTHESIS

input profiles, executing the proposed approach based on specific configurations, and logging the information for the analysis.

### B. Re-synthesizing Behavior

In this experiment, we focus on evaluating the ability of re-synthesizing process in dealing with synthesis failure. In this situation, it should compute the acceptable thresholds via Pareto set computation followed by re-synthesizing the model.

For the experiment, we apply three quality objective attributes (i.e. cost, time, and reliability) to simulate the quality requirements associated to an application. We then set the number of cloud collaborator, and the number of variation. Meanwhile, the quantitative information for each potential variation is generated randomly, within a range that can violate the satisfaction of multiple quality objectives.

We execute the decision-making process which fails at the first synthesis attempt. Then, the Pareto set computation is performed to obtain a set of acceptable thresholds which can potentially use to adjust the existing thresholds of each quality objective. One pair of the thresholds is selected randomly. Then, re-synthesis process is executed based on the adjusted thresholds. The relevant information is logged for the analysis and result. The simulations were performed on a virtual machine, CentOS 7, 10 GB memory with 8 CPUs.

The results in Table I shown the ability of re-synthesizing behavioral model by adjusting the initial thresholds of each quality objective. For instance, in the case of 10 potential cloud collaborators with 5 variations each, re-synthesis will be a success if the initial threshold set for $(cost, time, rel)$ as $[90.0, 853, 0.967]$ are adjusted to $[93.88, 1060, 0.978]$. The adjusted threshold set is randomly selected from many threshold sets. Meanwhile, the time spent to complete re-synthesis process is 6731ms. It is important to note that the presented values have been rounded to 2 decimal points for *cost* and 3 decimal points for *reliability*.

### C. Variability

In this experiment, we illustrate the benefit of capturing the potential collaborator in relation to the time variation. In general, having a higher number of time variation increases the number of potential collaborator. In addition, having more collaborator can improve the effectiveness of the decision. For this reason, we compare the mean potential collaborator or candidate in relation to the number of variation $[2....5]$.

For the experiment, we set similar quality objectives for each variation, where $cost < 90$, $time < 1000$, and $reliability > 0.9$. We also set similar number of cloud collaborators to 10, for each variation. For each variation, we fix the maximum utilization to 1, and randomize the current utilization value as well as the quantitative information related to the application. We set 100 of simulation cycle for each variation to obtain the mean value.

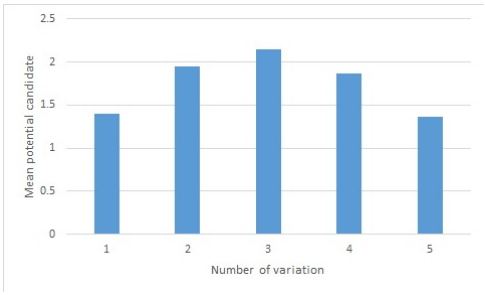


Fig. 6. Comparison of mean potential cloud collaborators with different number of variation

To collect the data, we execute the decision-making approach that includes, the encoding, the assignment and the synthesis process. This results in the winning strategies, also known as optimal plan. We then analyze this plan to identify the number of potential collaborator which can be chosen during the actual execution of autonomic cloud. The outcomes are shown in Figure 6. The simulations were performed on a virtual machine, Fedora 64-bit, 2 GB memory with a single CPU.

In the figure, we can see an increment mean of candidates that can be selected up to 3 variations, in particular, 1.4 mean value for 1 variation, 1.95 mean value for 2 variation and 2.15 mean value for 3 variation. However, the mean value decreases when the number of variation is kept increasing, while the number of available cloud collaborator remains with 10 collaborators. This pattern shows the important of having an optimal configuration between the number of available collaborator and its variation in order to improve the decision effectiveness.

### D. Scalability

In this experiment, we focus on illustrating the scalability of the approach, which is important towards improving its efficiency, especially for future work. For this reason, we compare its mean execution time during synthesis process by varying the number of cloud collaborator and workload variation.

We defined two set of configurations (A) and (B). For both configurations, we fix the quality objectives as $cost < 90$, $time < 1000$, and $reliability > 0.9$. We differentiate the configurations with different variation, namely, configuration (A) with 2 variation and configuration (B) with 4 variations. For each configuration, we set a number of cloud collaborator from 20 until 80. Meanwhile, we generate the quantitative information for each configuration randomly. We also set 100 for the simulation cycle of each combination of parameters, namely, the cloud collaborator and its variation.

Then, we execute the decision-making approach that includes, the encoding, the assignment and the synthesis process

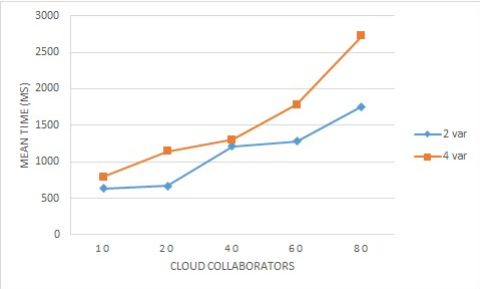


Fig. 7. Comparison of mean time between 2 and 4 variation over different number of cloud collaborator

based on the stated configurations. The assignment of values is performed during encoding process. In addition, there is no re-synthesizing process involved. We capture the execution time of the synthesis. The outcomes are shown in Figure 7. The simulations were performed on a virtual machine, Fedora 64-bit, 2 GB memory with a single CPU.

In overall, the results show that the mean time for (B) is taking longer time than (A). Specifically, for (B), it takes 0.9s higher than (A) with 80 cloud collaborators. Furthermore, the mean time between (A) and (B) has a closed distance when the collaborator is set to 40, with 0.093s.

The presented mean time is limited to the synthesis process, since this is the core task in the proposed approach. Having said that, the time is expected to increase if we consider other processes, especially re-synthesizing process. Furthermore, we expect the time taken will be also increased if the assignment is done as part of synthesis process. It is important to note that, even the higher variation takes longer time, it provides more options for achieving optimal decision. In conclusion, the scalability of the synthesis shows a reasonable time in respond to the increasing number of collaborator with 2 and 4 variations, when the collaborator reaches 80.

## VI. Related Work

In this section, we highlight the related works of the adaptation decision approaches for self-adaptation.

In Sykes et al [28], they addressed adaptation planning problem for assembling component configuration based on the utility objective function. The proposed two selection strategies. Firstly, the incremental selection which is less cost, but unable to guarantee an optimal section. Secondly, the aggregate selection which can find an optimal solution, but without considering the uncertainty factor.

Tajalli et al [29] proposed a plan-based approach that applies planning via model checking technique for generating adaptation plan during runtime. The approach takes the domain model specification and system goals as the inputs to generate the plan. However, this approach does not aims for generating an optimal plan with multiple quality objectives.

Coker et al [10] proposed the combination of probabilistic model checking and stochastic search techniques to address the optimal planning under uncertainty. These technique is is suitable to address a complex problem for searching an exponentially large space of candidate. However, the modeling aspect of the approach does not cater the crucial factors mentioned in this paper.

Moreno et al [23] addressed the limitation of reactive self-adaptation, by proposing a proactive approach using probabilistic model checking technique. The proposed decision approach captures the adaptation latency prediction and the uncertainty of the adaptation outcome in MDP model. The estimation of multiple quality attributes is based on the weighted aggregation function. Our work differ from this work, especially in the applied modeling technique and the strategy synthesis method. We use stochastic games to capture the uncontrolled environment which is needed to guarantee for optimal solution, especially under the worse case scenario. Furthermore, we formulate the quantitative reasoning as multi-objective properties synthesis that uses the Pareto-based computation.

The work by Camara et al [2] proposed an optimal planning approach under uncertainty with multiple quality objectives and latency factor via stochastic games model checking [6]. They used znn case study to illustrate the proposed approach. The estimation of quality attributes is transformed into the weighted aggregation function, which reduces multiple quality objectives as a single objective property synthesis. We distinguish our approach with this work in two aspects. Firstly, the modeling aspect where we capture the variability of the estimated quality attributes due to the workload variation of the cloud application deployment scenario. Secondly, the application of multi-objective strategy synthesis, instead of single objective, which can compute for Pareto-optimal solution without requiring the weight values.

In the context of autonomic cloud, Celestini et al [4], addressed the issue of selecting cloud collaborator where the resource information are not disclosed. They proposed a selection method by taking the reputation into consideration. The issues of optimality and uncertainty are beyond the scope of their work.

## VII. Conclusion

Effectiveness of self-adaptation decision mechanism is influenced by the flexibility that it can offer and its reliability of the outcome. One of the important supports when dealing with multiple quality objectives is to enable the computation even if some information i.e. weights and preferences are not provided. Furthermore, the modelling aspect to support the reasoning of the decision has to capture various information from the real-world scenario including the uncertainty and variability of the system environment context.

Therefore, we have proposed a self-adaptation decision approach based on the established stochastic games model checking technique. The approach covers the aspect of abstracting and encoding the decision-making behaviour and its uncertainty factor, as well as the variability of the quantitative information associated to the system environment, in particular, the autonomic cloud platform. In addition, the approach can automatically synthesize for the Pareto-optimal solution.

We have illustrated the decision effectiveness and performance of the approach by taking the cloud application deployment scenario into consideration. The results have shown the potential benefit of the approach to improve its effectiveness. Meanwhile, the presented performance behaviour illustrates a reasonable scalability of the proposed approach.

Future work will focus on validating the approach by integrating with a real-world environment setting. This is significance to generalize the adaptation effectiveness and efficiency in relation to the proposed approach. The outcome of this evaluation can be useful to realize runtime decision making approach for self-adaptive systems. Furthermore, the approach can be extended to handle collaborative adaptation with multiple autonomic managers. In the collaborative context, a cooperative decision making cannot be avoided. Thus, the issues such as inconsistency interaction has to be taken into consideration.

## ACKNOWLEDGMENT

A. Ismail acknowledges the previous support of Ministry of Higher Education Malaysia (MOHE) for Post-doctoral Scholarship and the support of Universiti Teknologi MARA, Selangor, Malaysia.

## REFERENCES

[1] J. Andersson, L. Baresi, N. Bencomo, R. de Lemos, A. Gorla, P. Inverardi, and T. Vogel. Software Engineering Processes for Self-Adaptive Systems. pages 51–75. Springer Berlin Heidelberg, 2013.

[2] J. Cámara, D. Garlan, B. Schmerl, and A. Pandey. Optimal planning for architecture-based self-adaptation via model checking of stochastic games. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC '15*, pages 428–435, New York, New York, USA, 2015. ACM Press.

[3] J. Cámara, G. A. Moreno, and D. Garlan. Reasoning about Human Participation in Self-Adaptive Systems. 2015.

[4] A. Celestini, A. Lluch Lafuente, P. Mayer, S. Sebastio, and F. Tiezzi. Reputation-Based Cooperation in the Clouds. In J. Zhou, N. Gal-Oz, J. Zhang, and E. Gudes, editors, *Trust Management VIII: 8th IFIP WG 11.11 International Conference, IFIPTM 2014, Singapore, July 7-10, 2014. Proceedings*, pages 213–220. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[5] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 8 2013.

[6] T. Chen, V. Forejt, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. On Stochastic Games with Multiple Objectives. pages 266–277. Springer, Berlin, Heidelberg, 2013.

[7] T. Chen, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. Synthesis for Multi-objective Stochastic Games: An Application to Autonomous Urban Driving. pages 322–337. Springer, Berlin, Heidelberg, 2013.

[8] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software Engineering for Self-Adaptive Systems: A Research Roadmap }Software Engineering for Self-Adaptive Systems. volume 5525, pages 1–26. Springer Berlin / Heidelberg, 2009.

[9] S.-W. Cheng, D. Garlan, and B. Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. pages 2–8. ACM, 2006.

[10] Z. Coker, D. Garlan, and C. Le Goues. SASS: Self-adaptation using stochastic search. 2015.

[11] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, and T. Vogel. Software engineering for self-adaptive systems: A second research roadmap. pages 1–32. Springer, 2013.

[12] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[13] A. Klarl, P. Mayer, and R. Hennicker. Helena@Work: Modeling the Science Cloud Platform. chapter Leveraging, pages 99–116. Springer Berlin Heidelberg, 2014.

[14] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 2014.

[15] M. Kwiatkowska, D. Parker, and C. Wiltsche. PRISM-games 2.0: A Tool for Multi-Objective Strategy Synthesis for Stochastic Games. In *TACAS*, 2016.

[16] M. Z. Kwiatkowska. Model Checking and Strategy Synthesis for Stochastic Games: From Theory to Practice (Invited Talk). In Y. R. Ioannis Chatzigiannakis Michael Mitzenmacher and D. Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:18, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[17] R. Laddaga. Guest Editor's Introduction: Creating Robust Software through Self-Adaptation. *IEEE Intelligent Systems*, 1999.

[18] R. Laddaga. Active software. pages 11–26. Springer, 2001.

[19] P. Mayer, A. Klarl, R. Hennicker, M. Puviani, F. Tiezzi, R. Pugliese, J. Keznikl, and T. Bure. The Autonomic Cloud: A Vision of Voluntary, Peer-2-Peer Cloud Computing. In *2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops*, pages 89–94. IEEE, 9 2013.

[20] P. Mayer, J. Velasco, A. Klarl, R. Hennicker, M. Puviani, F. Tiezzi, R. Pugliese, J. Keznikl, and T. Bureš. The Autonomic Cloud. In M. Wirsing, M. Hölzl, N. Koch, and P. Mayer, editors, *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*, pages 495–512. Springer International Publishing, Cham, 2015.

[21] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via VM multiplexing. In *Proceeding of the 7th international conference on Autonomic computing - ICAC '10*, page 11, New York, New York, USA, 2010. ACM Press.

[22] G. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Efficient Decision-Making under Uncertainty for Proactive Self-Adaptation. In *13th IEEE International Conference on Autonomic Computing (ICAC 2016)*, Würzburg, Germany, 2016.

[23] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, pages 1–12, New York, New York, USA, 2015. ACM Press.

[24] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):1–42, 5 2009.

[25] D. Serrano, S. Bouchenak, Y. Kouki, F. A. de Oliveira Jr., T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens. SLA guarantees for cloud services. *Future Generation Computer Systems*, 54:233–246, 2016.

[26] M. Svoreová and M. Kwiatkowska. Quantitative verification and strategy synthesis for stochastic games. *European Journal of Control*, 30:15–30, 2016.

[27] D. Sykes, W. Heaven, J. Magee, and J. Kramer. Exploiting non-functional preferences in architectural adaptation for self-managed systems. In *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*, pages 431–438, New York, New York, USA, 2010. ACM Press.

[28] D. Sykes, W. Heaven, J. Magee, and J. Kramer. Exploiting non-functional preferences in architectural adaptation for self-managed systems. pages 431–438. ACM, 2010.

[29] H. Tajalli, J. Garcia, G. Edwards, and N. Medvidovic. PLASMA: A Plan-based Layered Architecture for Software Model-driven Adaptation. In *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10*, page 467, New York, New York, USA, 2010. ACM Press.